Testing Plan

Glimpse Capstone Project

CAM SKUBIK-PEPLASKI and JOSHUA POOLE

Description of Overall Test Plan

The approach we plan to take for testing Glimpse will cover all major application functions. Each major component of the system will be tested individually, using a combination of automated and manual tests. A separate configuration will be created specifically for testing, which will run the automatic tests or aid the user in performing any semi-automatic tests. The back-end automated tests will be created using the Graphene testing tools, while the front-end tests will use the Jest testing framework and the DOM and React packages from the Testing Library. In addition, both the back-end and front-end will have manual tests performed. Finally, we will be testing the entire application as a whole by both developers and potential users after deployment.

Test Case Descriptions

1 SQLAlchemy Tests

SQLA1.1	**Seeding Test**
SQLA1.2	This test will ensure that the database can be seeded properly.
SQLA1.3	For this test we will run the back-end server in a testing environment and seed it with sample data. This will ensure the server is connected to the database properly. It will also test the creation of sample data using the SQLAlchemy models.
SQLA1.4	*Inputs*: Script that tells the application to run in testing mode and seed the database with specific sample data.
SQLA1.5	*Outputs*: Properly created sample data present in database.
SQLA1.6	Normal
SQLA1.7	Blackbox
SQLA1.8	Functional
SQLA1.9	Unit

SQLA2.1	**Reading Test**
SQLA2.2	This test will ensure that the database can be queried using SQLAlchemy model query methods.
SQLA2.3	For this test we will run the back-end server and database in a testing environment. The database will be seeded with sample data, then unit tests will be run that query the database using the SQLAlchemy model methods.
SQLA2.4	*Inputs*: Query request to database from SQLAlchemy models.
SQLA2.5	*Outputs*: Queried objects returned from database correctly.
SQLA2.6	Normal
SQLA2.7	Blackbox
SQLA2.8	Functional
SQLA2.9	Unit

SQLA3.1	**Creating Test**
SQLA3.2	This test will ensure that new entries can be created properly in the database though the SQLAlchemy model create methods.
SQLA3.3	For this test we will run the back-end server and database in a testing environment. The unit tests will be run, and the each model's create methods will be called. Sample data be passed to each of these methods which will create a new entry in the database for each table.
SQLA3.4	*Inputs*: Create requests to the database for each SQLAlchemy model. Sample inputs for each create method.
SQLA3.5	*Outputs*: Confirmation of successful database entry creation and newly created objects displayed.
SQLA3.6	Normal
SQLA3.7	Blackbox
SQLA3.8	Functional
SQLA3.9	Unit

SQLA4.1	**Updating Test**
SQLA4.2	This test will ensure that entries in the database can be updated by using the SQLAlchemy model update methods.
SQLA4.3	For this test we will run the back end-server and database in a testing environment. The database will be seeded with sample data, then each model's update method will be called. Sample data will be passed to each method, which will update the seeded data.
SQLA4.4	*Inputs*: Update request to the database for each SQLAlchemy model's respective sample data. Sample inputs for each update method.
SQLA4.4	*Outputs*: Confirmation of successful updating of each database entry and newly updated object displayed.
SQLA4.6	Normal
SQLA4.7	Blackbox
SQLA4.8	Functional
SQLA4.9	Unit

SQLA5.1	**Deleting Test**
SQLA5.2	This test will ensure that entries in the database can be deleted using the SQLAlchemy model delete methods.
SQLA5.3	For this test we will run the back-end server and database in a testing environment. The database will be seeded with sample data, then each model's delete method will be called. Identifying information for each sample data entry in the database will be passed to the respective delete method, which will delete the entry in the database.
SQLA5.4	*Inputs*: Delete requests to the database for each SQLAlchemy model's respective sample data. Identifying information required to delete a specific entry.
SQLA5.4	*Outputs*: Confirmation of successful deletion of each sample database entry.
SQLA5.6	Normal
SQLA5.7	Blackbox
SQLA5.8	Functional
SQLA5.9	Unit

2 GraphQL and Graphene Tests

GQL1.1	**Client Reading Test**
GQL1.2	This test will ensure that the client is able to query and receive database records from the back-end.
GQL1.3	This test will attempt to read the same data as SQLA2, but using a query made available to the front-end client.
GQL1.4	*Inputs*: Query request to back-end from front-end application.
GQL1.5	*Outputs*: The correct sample data present in database, in a form consumable by the client.
GQL1.6	Normal
GQL1.7	Blackbox
GQL1.8	Functional
GQL1.9	Integration

GQL2.1	**Client Creating Test**
GQL2.2	This test will ensure that the client is able to create database records via a request sent to the back-end.
GQL2.3	This test will attempt to create the same record as SQLA3, but using a mutation made available to the front-end client.
GQL2.4	*Inputs*: Create request to back-end from front-end application, including the data to create a record with.
GQL2.5	*Outputs*: Either the created record as given, in a form consumable by the client, or a boolean indicating success/fail.
GQL2.6	Normal
GQL2.7	Blackbox
GQL2.8	Functional
GQL2.9	Integration

GQL3.1	**Client Updating Test**
GQL3.2	This test will ensure that the client is able to update database records via a request sent to the back-end.
GQL3.3	This test will attempt to update the same record as SQLA4, but using a mutation made available to the front-end client.
GQL3.4	*Inputs*: Update request to back-end from front-end application, including an identifier for a record and the data needed to modify it.
GQL3.5	*Outputs*: Either the updated record with the correct modifications, in a form consumable by the client, or a boolean indicating success/fail.
GQL3.6	Normal
GQL3.7	Blackbox
GQL3.8	Functional
GQL3.9	Integration

GQL4.1	**Client Deleting Test**
GQL4.2	This test will ensure that the client is able to delete database records via a request sent to the back-end.
GQL4.3	This test will attempt to delete the same record as SQLA5, but using a mutation made available to the front-end client.
GQL4.4	*Inputs*: Delete request to back-end from front-end application, including an identifier for a record.
GQL4.5	*Outputs*: A boolean indicating success/fail.
GQL4.6	Normal
GQL4.7	Blackbox
GQL4.8	Functional
GQL4.9	Integration

3 API | Application Programming Interface Tests

API1.1	**API Client-Server Connection**
API1.2	This test will ensure that the web client can communicate with the back-end server through the API.
API1.3	This test will involve a simple GET request to a test endpoint. The client will receive a response from the test endpoint that signifies a successful connection.
API1.4	*Inputs*: Http request from the client a to test endpoint.
API1.4	*Outputs*: Http response from the server to the client signifying a successful connection.
API1.6	Normal
API1.7	Blackbox
API1.8	Functional
API1.9	Integration

API2.1	**API Model Upload**
API2.2	This test will ensure that the model upload endpoint works correctly.
API2.3	A sample model file will be sent the the model upload API endpoint through a http request. The back-end server will then create a new blob in Azure Storage and upload the model file to it. The blob URL for the model will then be sent back to the client through an http response.
API2.4	*Inputs*: User input and sample 3D model file.
API2.4	*Outputs*: Successful delivery of model file to the back-end and successful upload of model to a blob in Azure storage. Additionally the blob URL where the model will be stored.
API2.6	Normal
API2.7	Blackbox
API2.8	Functional
API2.9	Unit

4 CF | Client Functionality Tests

CF1.1	**User Log-in**
CF1.2	This test will ensure that a user is able to create an account and log into the website using their newly created account.
CF1.3	The user will navigate to the "Login" page and register a new account. They will enter their account information and click register. They will then be redirected to the website splash page on the newly created account.
CF1.4	*Inputs*: User mouse input and information required to register a new account.
CF1.4	*Outputs*: Newly created user entry in database and website splash page.
CF1.6	Normal
CF1.7	Blackbox
CF1.8	Functional
CF1.9	Integration

CF2.1	**Client Model Upload**
CF2.2	This test will ensure that a user can view and uploaded model.
CF2.3	The user will navigate to the login page, enter their credentials and login. After landing at the splash page after login, they will press the upload button and select a file from their local machine to be uploaded to the website. After uploading the file, they will travel to their account page and where their model can be viewed.
CF2.4	*Inputs*: User mouse and keyboard input and a model file from the user's local machine.
CF2.4	*Outputs*: Correct website response from user interactions and display of newly uploaded model.
CF2.6	Normal
CF2.7	Blackbox
CF2.8	Functional
CF2.9	Integration

CF3.1	**3D Model Viewing**
CF3.2	This test will ensure that the client 3D model viewer works correctly and a user can interact with it.
CF3.3	The user will navigate to the login page, enter their credentials and login. After landing at the splash page after login, they will click on any of the public models on the page. After traveling to the specific model's viewer page, they will test the transformation widgets available for the model such as rotate and zoom.
CF3.4	*Inputs*: User keyboard and mouse input for login and model viewer controls.
CF3.4	*Outputs*: Display of model and correct transformation based on controls interacted with.
CF3.6	Normal
CF3.7	Blackbox
CF3.8	Functional
CF3.9	Integration

5 Jest Tests

J1.1	**Home Page Rendering**
J1.2	This test will ensure that the correct home page components are being rendered as expected.
J1.3	This test will utilize Jest tests to check that each component necessary for the homepage to function correctly is present after the page renders. It will also ensure the components function as expected where such testing is possible.
J1.4	*Inputs*: N/A, although a test file must be written.
J1.4	*Outputs*: A pass or fail verdict, along with any exceptions or errors that may have occurred.
J1.6	Normal
J1.7	Blackbox
J1.8	Functional
J1.9	Unit

J2.1	**3D Viewer Rendering**
J2.2	This test will ensure that the correct components are being rendered in the 3D model viewer as expected.
J2.3	This test will utilize Jest tests to check that each component necessary for the 3D viewer to function correctly is present after the page renders. It will also ensure the components function as expected where such testing is possible.
J2.4	*Inputs*: N/A, although a test file must be written.
J2.4	*Outputs*: A pass or fail verdict, along with any exceptions or errors that may have occurred.
J2.6	Normal
J2.7	Blackbox
J2.8	Functional
J2.9	Unit

J3.1	**Model Upload Rendering**
J3.2	This test will ensure that the correct model upload page components are being rendered as expected.
J3.3	This test will utilize Jest tests to check that each component necessary for model uploading to function correctly is present after the page renders. It will also ensure the components function as expected where such testing is possible.
J3.4	*Inputs*: N/A, although a test file must be written.
J3.4	*Outputs*: A pass or fail verdict, along with any exceptions or errors that may have occurred.
J3.6	Normal
J3.7	Blackbox
J3.8	Functional
J3.9	Unit

J4.1	**Model Details Rendering**
J4.2	This test will ensure that the correct model details configuration page components are being rendered as expected.
J4.3	This test will utilize Jest tests to check that each component necessary for the configuration of model details to function correctly is present after the page renders. It will also ensure the components function as expected where such testing is possible.
J4.4	*Inputs*: N/A, although a test file must be written.
J4.4	*Outputs*: A pass or fail verdict, along with any exceptions or errors that may have occurred.
J4.6	Normal
J4.7	Blackbox
J4.8	Functional
J4.9	Unit

J5.1	**Model Modal Rendering**
J5.2	This test will ensure that the correct components for the model modal are being rendered as expected.
J5.3	This test will utilize Jest tests to check that each component necessary for the model viewing modal to function correctly is present after the page renders. It will also ensure the components function as expected where such testing is possible.
J5.4	*Inputs*: N/A, although a test file must be written.
J5.4	*Outputs*: A pass or fail verdict, along with any exceptions or errors that may have occurred.
J5.6	Normal
J5.7	Blackbox
J5.8	Functional
J5.9	Unit

6 Web Server Tests

WS1.1	**Azure Deployment**
WS1.2	This test ensures that the website and all required components can be successfully deployed to Microsoft Azure.
WS1.3	A staging version of the application will be manually deployed to a combination of Azure services specifically created to accept it. The website will then be interacted with from the UI, ensuring everything operates as intended.
WS1.4	*Inputs*: Deployable images of the application in a staging environment and user input on the remote website.
WS1.4	*Outputs*: Confirmation of successful deployment and correct responses to user interaction.
WS1.6	Normal
WS1.7	Whitebox
WS1.8	Performance
WS1.9	Integration

7 Full System Tests

FS1.1	Full System User Testing
FS1.2	This test will ensure the application, once deployed, can support longer-term usage by multiple people.
FS1.3	A handful of users will work with the application both simultaneously and periodically over a longer period of time. They will go through various actions within the client to make sure it all functions correctly and the application can handle a larger number of connections at once.
FS1.4	*Inputs*: 3+ active connections, performing manual tests and other actions. These should be both concurrent and intermittent over the course of the test.
FS1.4	*Outputs*: Any errors, bugs, or exceptions encountered by users or that occur on the application's host platform.
FS1.6	Normal
FS1.7	Blackbox
FS1.8	Performance
FS1.9	Integration

Test Case Matrix

ID	Normal/ Abnormal/ Boundary	Blackbox/ Whitebox	Functional/ Performance	Unit/ Integration
SQLA1	Normal	Blackbox	Functional	Unit
SQLA2	Normal	Blackbox	Functional	Unit
SQLA3	Normal	Blackbox	Functional	Unit
SQLA4	Normal	Blackbox	Functional	Unit
SQLA5	Normal	Blackbox	Functional	Unit
GQL1	Normal	Blackbox	Functional	Integration
GQL2	Normal	Blackbox	Functional	Integration
GQL3	Normal	Blackbox	Functional	Integration
GQL4	Normal	Blackbox	Functional	Integration
API1	Normal	Blackbox	Functional	Integration
API2	Normal	Blackbox	Functional	Unit
CF1	Normal	Blackbox	Functional	Integration
CF2	Normal	Blackbox	Functional	Integration
CF3	Normal	Blackbox	Functional	Integration
J1	Normal	Blackbox	Functional	Unit
J2	Normal	Blackbox	Functional	Unit
J3	Normal	Blackbox	Functional	Unit
J4	Normal	Blackbox	Functional	Unit
J5	Normal	Blackbox	Functional	Unit
WS1	Normal	Whitebox	Performance	Integration
FS1	Normal	Blackbox	Performance	Integration